|  |  |
| --- | --- |
| { **name**: string, **id**: number,  **income**: number, **rank**: number,  **major**: string,  **year**: string,  **registrationDate**: **ISODate time**,  **enrolled**: **array** of **numbers** of enrolled classes,  **grades**: [ **array** of sub-documents,  { **course**: course **number**,  **grade**: **number** of grade } ]  } | { **name**: 'Agamemnon', **id**: 19,  **income**: 81800, **rank**: 12,  **major**: "Ventriloquism",  **year**: '2-Sophomore',  **registrationDate:** **2017-07-09T16:17:35,**  **enrolled**: [204, 202, 201],  **grades**: [ { **course**: 102, **grade**: 79 },  { **course**: 103, **grade**: 74 },  { **course**: 101, **grade**: 94 } ]  } |
| [Mongo commands](https://github.com/EdgeTechAcademy/MongoDB_Files/blob/master/students.mongo) to build the collection in the examples | |

| **Query Operators** | |
| --- | --- |
| **db.students.find({query}, {projection})** | **Docs matching where** |
| { **name**: 'Umberto' } | **name** is 'Umberto'. True if field or array contains 'Umberto'? |
| { **name**: 'Umberto', **enrolled**: 301 } | **name** is "Umberto" and is **enrolled** in 301 |
| { **enrolled**: { $gt: 403 } } | **enrolled** is greater than 403.  $eq (=), $gte (>=) , $lt (<) , $lte (<=) , and $ne (!=) |
| { **enrolled**: { $size: 5 } } | **enrolled** array has 5 entries |
| { **name**: { $in: ['Umberto', "Veronica"] } } | **name** is either 'Umberto' or "Veronica". |
| { **enrolled**: { $all: [401, 402, 405] } } | **enrolled** contains all codes (401, 402 and 405) |
| { "**grades.grade**": { $lte: 69 } } | embedded document **grades** has **grade** <= 69 |
| { **grades**: { $elemMatch:  { **course**: 103, **grade**: 29 } } } | **grades** array that contains an element with both  **course** equal to 103 and grade equal to 29. |
| { **grades**: { $elemMatch:  { **course**: 103, **grade**: { $lte: 69 } } } } | Students that failed course 103 |
| {$and: [ {**enrolled**: 103 } , {**year**:'1-Freshman' }]}  Same as  { **enrolled**: 103, **year**: '1-Freshman' } | **All** boolean expressions in array are True: students are Freshmen and enrolled in course 103.  This is the same as this simple query  $not can be used to reverse the result |
| { $and: [ { **enrolled**: 305 } ,  { $or: [ { **year**: '3-Junior' } ,  {**year**: '4-Senior' } ] } ] } | **At least one** boolean expression is True  Jrs **or** Srs **enrolled** in course 305 |
| { **name**: /^M/ }  { **name**: /M$/ } | **name** begins(^) / ends($) with the 'M'.  also use the regex operator: { a: { $regex: "^m" } } |
| { **"grades.grade"**: { $mod: [10, 1] } } | **Grades** ending in 1 (61, 71, 81, …). |
| { **online**: { $exists: true } } | Returns docs if the existence of the field '**online'** matches the predicate. (Returns docs if online exists in a document) |
| { **name**: { $type: 2 } } | **name** is a string. See bsonspec.org for more. |
| { $text: { $search: "hello" } } | Docs that contain "hello" on a text search. **Requires a text index.** |
| { major: { $nin: ["Tennis", "Jacks"] } } | Major is anything but 'Tennis' or "Jacks". |
| { grades: { $size: 3 } } | Grades has exactly 3 elements. |
| {}, { \_id: 0, **name**: 1, **enrolled**: 1 } | **Projection.** Reshape results with specified fields. \_id is always on (\_id:0 to remove). Turn off individual fields using **field:0** |
| {}, {enrolled: { $slice: 2 } } | **Projection.** Return the first two classes in the enrolled array, (-2 will return the last two classes |
| {}, { grades: { $slice: [-5, 3] } } | **Projection.** Return 3 grades beginning with the 5th item from the end |

|  |  |
| --- | --- |
| Update Operators | |
| db.students.update({query},{update},{options}) | **Options: {upsert: true/false, multi: true/false}** |
| {name: 'Franklin' }, { $set: { name: 'Adams' } } | Rename a student |
| { grades: { $elemMatch:  { course: 103, grade: { $eq: 55 } } } },  { $set: { " grades.1.grade ": 99 } } | Find document with enrolled course = 103 and grade = 55.  Update the grade for the **2nd** course to 99 |
| { $unset: { deansList: 1 } } | Removes the **deansList** field from the document. |
| {}, { $max: { tuition: 5000 } }, {multi:true}  {name: 'Adams' }, { $min: { tuition: 0 } } | Sets tuition to the greater value, either current or 5000. If tuition does not exist sets tuition to 5000.  **$min** does what you think it would |
| { grades: { $elemMatch:  { course: 103, grade: { $lte: 69 } } } },  { $inc: { "grades.$.grade": 10 } } | Was the final in 103 too hard and you want to give everyone (use updateMany) an extra 10 points? If you have taken the class twice only the first one will be updated ("Agamemnon") |
| { $mul: { tuition: 0.5 } } Half scholarship | Sets tuition to (current value x 0.5). If tuition does not exist sets tuition to 0. |
| {}, {$rename: {"classes": "enrolled"}}, {multi:true} | Renames field classes to course. And do it for all documents. |
| {},{$setOnInsert: {registered: 1} } ,{ upsert: true } | Sets field **registered** to 1 in case of upsert operation. |
| {}, { $currentDate: { startDate: { $type: "date" } } } | Sets field **startDate** with the current date.  $currentDate can be specified as date or timestamp. |
| {}, { $bit: { flags: { and: 7 } } } | bitwise operation over a field. Supports **and**|**xor**|**or**  If flag is 12: 1100 && 0111 🡺 0100 |

| **Array Update Operators** | |
| --- | --- |
| db.students.**update**() |  |
| {}, { $push: { a: 1 } } | Appends the value 1 to the array a. |
| {}, { $push: { a: { $each: [2, 2, 3, 3, 4, 4] } } } | Appends both 2, 2, 3, 3 and 4, 4 to the array a. |
| {}, { $addTo**Set**: { a: 5 } } | Appends the value 5 to the array a ( if the value doesn't already exist ) . |
| {}, { $addTo**Set**: { a: {$each: [5, 6]} }} | Appends both 5 and 6 to the array a ( if they don't already exist ) . |
| {}, { $pop: { a: 1 } } | Removes the **last** element from the array a. |
| {}, { $pop: { a: -1 } } | Removes the **first** element from the array a. |
| {}, { $pull: { a: 2 } } | Removes all occurrences of 5 from the array a. |
| {}, { $pullAll: { a: [3, 4] } } | Removes multiple occurrences of 5 or 6 from the array a. |

| **Aggregation Pipeline Operators** | |
| --- | --- |
| db.students.**aggregate()** | **filter** **operations** the number of documents passed along |
| { $match: { enrolled: 205 } } | Passes only documents where enrolled = 205. |
| { $match: { grades: { $elemMatch: { course: 103, grade: { $lte: 89 } } } } } | Add the usual filter expressions in here as well |
| { $limit: 10 } | Limits the set of documents to 10, passing the first 10 documents. |
| { $sort: { name:1 } } | Sorts results by field name ascending. |
| { $skip: 10 } | Skips the first 10 documents and passes the rest. |
| { $skip: 5 } , { $limit: 4 } | Returns documents 6,7,8,9 |
| { $sample: { size: 3 } } | Randomly selects the specified number of documents |

| **Pipeline Operator** **- multiplies** the number of documents passed along the pipeline | |
| --- | --- |
| { $unwind: "$grades" }  , { $project: { \_id: 0, name: 1, grades: 1 } }  =-=-=-=-= before =-=-=-=-=  **"grades" : [ { "course" : 103, "grade" : 80 }**, **{ "course" : 104, "grade" : 83 }**, **{ "course" : 105, "grade" : 99 } ]** | Deconstructs grades array into individual docs of each element. If the array has 5 entries the original document will have an new field added with the item from the array  =-=-=-=-= $unwind =-=-=-=-=  **"grades" :** **{ "course" : 103, "grade" : 80 },**  **"grades" : { "course" : 104, "grade" : 83 },**  **"grades" : { "course" : 105, "grade" : 99 }** |
| {$unwind: {path: "$grades", includeArrayIndex:"**idx**" } }  , { $project: { \_id: 0, name: 1, grades: 1, **idx**: 1 } } | "grades" : { "course" : 103, "grade" : 80 }, "**idx**" : NumberLong(0) }  "grades" : { "course" : 104, "grade" : 83 }, "**idx**" : NumberLong(1) }  "grades" : { "course" : 105, "grade" : 57 }, "**idx**" : NumberLong(2) } |
| Pipeline Operators that **reshape** the output (limits or changes or adds fields) | |
| { $project: { \_id: 0, name: 1, grades: 1 } } | Reshapes output to include only field name and grades, removing others. \_id defaults to visible, 0 switches it off. |
| { $project: { Student:"$name", grades: 1 } } | Reshapes output to include only \_id, grades and the new field Student with the value of name |
| { $project: { Rank2: { $add:["$rank", 10] } } } | Reshapes output to include only \_id and Rank2, set to the sum of rank + 10. (Also $subtract, $multiply, $divide, $mod) |
| { $sort: { year: -1, "grades.grade": 1 } }  , {$project:{ \_id:0, name: 1, year:1 } } | Sort by Year descending and grades (1st grade in array of grades) ascending. |
| { $unwind:"$grades" }  , { $lookup: { from: "courses"  , localField: "grades.course"  , foreignField: "course"  , as: "CourseName" } } | Look up the course number in the course catalog, 'join' on course field, return the **array** of documents that match and append to the result stream. See Examples page |
| { $group: { \_id: "$major", count: { $sum:1 }}}  { $group: { \_id: "$year", count: { $sum:1 }}} | Groups documents by field (major, year) and counts each distinct a value. |
| **{ $group: { \_id: "$major",**  **major: { $addToSet: "$year" } } }**  **{ $group: { \_id: "$major",**  **year: { $addToSet: "$year" } } }** | Groups documents by field (major/year) with new field consisting of a **set** of values from year/major. |
| { $out: "**results**" } | Writes resulting documents in the pipeline into the collection "**results**". Must be the last stage of the pipeline. |
| { $facet: {  **"byClass":** [ { **$bucketAuto**: {  groupBy: "$**year**", buckets: 5,  output: { "count": { $**sum**: 1 },  "students": { **$push:** **"$name"** } } } } ],  **"byMajor":** [ { **$bucketAuto**: {  groupBy: "$**major**", buckets: 7,  output: { "count": { $**sum**: 1 },  "students": { **$push**: { "name": "$name",  "year": "$year" } } } } } ]  } } | Facet sends a document stream to multiple pipelines to analyze different facets of the stream. The output stream is replaced by the results of the facets.  **"byClass":** groups documents by year (Fresh, Soph,…), builds buckets for each year, counts the entries and pushes the students names to an array.  **"byMajor"** groups documents by major, builds buckets for each major, counts the entries and pushes a document of student name and year into an array for each bucket.  **$bucketAuto:** you select the number of buckets |
| { $facet: { **"Income"**: [ { $**bucket**: {  groupBy: "$**income**",  boundaries: [ 0, 20000, 40000, 60000, 80000 ],  default: "80K+",  output: { "count": { $sum: 1 },  "students": **{ $push**: { "income": "$income",  "major": "$major" } } } } } ]  } } | **$bucket:** you define the bucket ranges, including a default for documents above and below the defined ranges.  Create buckets based on income  >= 0 to < 20K >= 20K to < 40K >- 40K to < 60K  >= 40 to < 80K >= 80K **and** < 0  Output will have a count field and a sub-doc with income and major |
| { **$unwind**: "$grades" },  { **$sortByCount**: "$grades.grade" }  { **$unwind**: "$grades" },  { **$sortByCount**: "$grades.course" }  { **$sortByCount**: "$major" }  { **$sortByCount**: "$year" } | **$sortByCount:** analysis on stream results sorted by count. Show the top scores  Shows the most taken course  Majors by popularity  Year in school |
| { $sort: { name: -1, rank: 1 } }  , { $project:{ \_id: 0, name: 1, rank:1 } } | Sorts documents |
| { $match: { "grades.grade": { $gt: 90 } } },   { **$count**: "'A' Students" } | Counts items in pipeline  Filter out low scores, count the remaining records |
| { $**addFields**: {  totalHomework: { $**sum**: "$homework" }  , totalQuiz: { $**sum**: "$quizzes" }  , extraCredit: { $**sum**: "$extra" } } } | Use addFields to create new fields in the stream.  **Sum** the scores for each student in each course  Uses the **scores collection**  , { $project: { \_id: 0, name: 1, totalHomework: 1  , totalQuiz: 1, extraCredit: 1 } } |
| , { $**addFields**: { **final**: {  student: "$student"  , course: "$course"  , "Homework @ 60%": "$totalHomework"  , "Quizzes @ 40%": "$totalQuiz"  , extraCredit: "$extraCredit"  , totalScore: {  $add:[  { $multiply: ["$totalHomework", 0.4 ] }  , { $multiply: ["$totalQuiz", 0.6 ] }  , "$extraCredit" ] } } } } | Use addFields to create a new subdocument **final**, with the student information and scores. The totalHomework and totalQuiz scores are weighted and all weighted scores are summed together. |
| , { $**replaceRoot**: { newRoot: "$**final**" } } | Remove details. Set root to final scores sub-document  This is also in the examples page |

| **Aggregate Set Operations** | |
| --- | --- |
| { $project: { A:1, B:1, SET, \_id: 0 } }  sameElements: { **$setEquals**: [ "$A", "$B" ] }  AisSubset: { **$setIsSubset**: [ "$A", "$B" ] } | Set comparison operators – adds new Boolean field  All elements in A are in B and all in B are in A  All elements in A are in B |
| { $project: { A:1, B:1, SET, \_id: 0 } }  allValues: { $setUnion: [ "$A", "$B" ] }  common: { $setIntersection: [ "$A", "$B" ] }  inAOnly: { $setDifference: [ "$A", "$B" ] } | Set combination operators – adds new set field  Include all values from both sets  Only values common in both sets  Remove any set B items from set A |
| { $project: { A:1, B:1, SET, \_id: 0 } }  anyTrue: { $anyElementTrue: [ "$answers" ] } allTrue: { $allElementsTrue: [ "$answers" ] } | True is anything besides: 0, false, null, [] |
| **Comparison Aggregate Operations** | |
| lastYear: { **$eq**: [ "$year", "4-Senior" ] }  top10: { **$lte**: [ "$rankyear", 10 ] }  incomeTarget: { **$cmp**: [ "$income", 50000 ] } | Creates a new field in the output stream  $gt, $gte, $lt, $lte, $eq, $ne (all return T/F), boolean field  $cmp returns number -1 if less than, 0 if equal, 1 if greater than |

|  |  |
| --- | --- |
| **Arithmetic Aggregation Operators** | |
| {$project:{delta:{ $abs: { $subtract: [ 10, 99 ] } }}} | absolute value of a number. (i.e. 89) |
| {$project:{billingDate: {$add: [ "$registrationDate",  3\*24\*60\*60**000** ]}}} | If one element is a date, date arithmetic is performed assuming **milliseconds** |
| {$project:{ceilingValue: { $ceil: 7.00001 } }} | smallest int >= to the number. (i.e. 8) |
| {$project:{workdays: { $divide: [ "$hours", 8 ] }}} | A/B |
| {$project:{toE: { $exp: 2}}} | *e*X (in this case e2 = 7.38905609) |
| {$project:{intValue: { $floor: 8.999999999999 }}} | largest int <= to the number. (i.e. 8) |
| {$project:{logOfSales: { $ln: "$sales" }}} | the natural log |
| {$project:{logOfNum: { $log: [ 32, 2 ] }}} | the log of a number in the specified base. [#, base] (i.e. 5, 25 = 32) |
| {$project:{Log10: { $log10: 1000000 }}} | the log base 10 of a number. (i.e. 6, 106) |
| {$project:{minutes: { $mod: [ "$duration", 60 ] }}} | remainder of the first number divided by the second. |
| {$project:{total: { $multiply: [ "$price", "$quantity" ] }}} | A\*B\*C\*… |
| {$project:{variance: { $pow: [{$stdDevPop:"$homework" },2] }}} | Raises a number to the specified exponent. |
| {$project:{distance: { $sqrt: {  $add: [ { $pow: [ { $subtract: [ "$p2.y", "$p1.y" ] }, 2 ] },  { $pow: [ { $subtract: [ "$p2.x", "$p1.x" ] }, 2 ] } ]  } }}} | Calculates the square root.  sqrt(A2 + B2) |
| {$project:{dateDifference: { $subtract:[ "$date", 5\*60\*1**000** ] }}} | Subtract two numbers if one is a date then do date arithmetic assuming number is milliseconds |
| {$project:{intValue: { $trunc: "$value" }}} | Truncates a number to its integer. |

|  |  |
| --- | --- |
| Pipeline String Operations that **manipulate text** | |
| {$project: {test:{ $concat:["$name", " - ", "$year"]}}} | Concatenate any number of strings  (**students** collection) |
| { $strcasecmp:["$quarter", "17q4"]} | Case-insensitive comparison |
| { $substrBytes:["Hello World!", 6, 5  ]} | "World" ["String", start at, ends before]. Same for CP |
| item: { $toUpper: "$item" } $toLower | Guess what they do |
| fullName: { $concat: [ "$last", ", ", "$first" ] } | Concatenates multiple strings together |
| { $indexOfBytes: ["cafeteria", "e"] }  { $indexOfCP: ["cafétéria", "**é**"] }  { $indexOfBytes: ["cafétéria", "e"] } | 3 results are the same for the Bytes vs. the  3 CP (CodePoint) versions of the operator  -1 |
| {$project: {parts: { $split: [ "July-20-1969", "-" ] }}}) | [ "July", "20", "1969" ], split string based on a delimiter |
| $strLenBytes | Returns the length of the field. Also CP |

|  |  |
| --- | --- |
| **Aggregation Variable Operators** | |
| { $project: { \_id:0, name:1, totalSalary: {  **$let**: {  vars: { salary: { $add: [ '$income', 1000 ] },  premium: { $cond:  { if: { $eq: ['$major', 'Aerialist']}, then: 1.10, else: 1 } }  },  in: { $multiply: ["$$salary", "$$premium"] }  } } } } | $let Defines variables for use within the scope of a subexpression and returns the result of the subexpression.  Here we give an increase in salary based on major.  To clean up the extra decimals  ,{$project: {name:1, totalSalary: {$trunc:"$totalSalary"} }}  (**students** collection) |
| { $project: { \_id:0, comment: { $literal: "future use" } } } | Return a value without parsing. Use for values that the aggregation pipeline may interpret as an expression. |

|  |  |
| --- | --- |
| **Array Aggregation Operators** | |
| { $project: {last: { $arrayElemAt: [ "$homework", 0 ]}}}  { $project: {first: { $arrayElemAt: [ "$quizzes", -2 ] }}} | Returns element at array index. (**scores** collection)  0 -- 1st item, -1 -- last, -2 -- 2nd to the last |
| { $project:{items: { $concatArrays: [ "$homework", "$quizzes" ] }}} | Concatenates arrays to return the concatenated array.  (**scores**) |
| {$project: { \_id: 0, name: 1, failing: {  $filter: {  input: "$grades",  as: "grade",  cond: { $lt: [ "$$grade.grade", 70 ] }  } } } } | Selects a subset of the array to return an array with only the elements that match the filter condition.  Take the grades array and filter **out** elements >= 70.  (**students**) |
| { $project: { \_id:0, name:1, "taking202" : {  $in: [ 202, "$grades.course" ] } } } | **Boolean** indicating whether a specified value is in an array  (**students**) |
| { $project: { **index**: {$indexOfArray: ["$quizzes", 100 ] }}}  , { $match: { **index**: { $gt: -1 } } } | Searches an array for a value and **returns** index of the first occurrence. If not found, returns -1. (**scores**) |
| { $project: { "student" : "$student",  " Perfect Score" : { $in: [ 100, "$homework" ]}}} | **Boolean** indicating whether a specified value is in an array.  (**scores**) |
| $isArray | Determines if the operand is an array. Returns a boolean. |
| { $project: { \_id:0, student:1, adjustedGrades: {  $map: {  input: "$quizzes",  as: "grade",  in: { $add: [ "$$grade", 2 ] }  } } } } | Applies a subexpression to each element of an array and returns the array of resulting values in order. Accepts named parameters.  Add 2 points to everyone's quizzes  (**scores**) |
| { $project: { range: { $range : [ -12, 5, 3 ] }}} | Outputs an array containing a sequence of integers  [start, end, increment] |
| $reverseArray | Returns an array with the elements in reverse order. |
| { $project: { "\_id": 0, "student": 1, "course": 1,  "quizzes": { $reduce: {  input: "$quizzes",  initialValue: 0,  in: { $add: ["$$value", "$$this"] }  } }  , "extraCredit": { $reduce: {  input: "$extra",  initialValue: 0,  in: { $add: ["$$value", "$$this"] }  } }  , "homework": { $reduce: {  input: "$homework",  initialValue: 0,  in: { $add: ["$$value", "$$this"] }  } } } } | Applies an expression to each element in an array and combines them into a single value.  {"student": 1, "course": 204, "quizzes": 278, "extraCredit": 7}  {"student": 1, "course": 202, "quizzes": 278, "extraCredit": 4}  {"student": 1, "course": 201, "quizzes": 116, "extraCredit": 9}  (**scores**) |
| $size | Returns the number of elements in the array. |
| { $project: { \_id: 0, student: 1,  first: { $slice: ["$homework", 1] },  last2: { $slice: ["$homework", -2, 2] } } } | Returns a subset of an array.  1 is the first element, -1 is the last  [array, position, count] (**scores**) |
| $zip | Merge two lists together. |

|  |  |
| --- | --- |
| **Date** Aggregation Operators | |
| { $project: {  year: { $year: "$registrationDate" },  hours: { $hour: "$registrationDate" },  dayOfYear: { $dayOfYear: "$registrationDate" },  dayOfWeek: { $dayOfWeek: "$registrationDate" },  week: { $week: "$registrationDate" }  } } | All values are returned as numbers  Year: 4 digit, $**month**: 1 – 12, $**dayOfMonth**: 1 – 31  Hours: 0 – 23, $**minute**: 0 – 59, $**second**: 0 – 60 (leap))  1 to 366  1 (Sunday) and 7 (Saturday)  0 partial week before 1st Sunday to 53 (leap year)  (**students**) |

|  |  |
| --- | --- |
| **Conditional** Aggregation Operators | |
| **{ $project: { \_id:0, name:1, income:1,** discount:{ **$cond:**  { if: { $lte: [ "$income", 250000 ] }, then: .30, else: 0 } } **} }**  Or  **{ $project: { \_id:0, name:1, income:1,** discount: { **$cond:**  [ { $lte: [ "$income", 250000 ] }, .30, 0 ] } **} }** | ternary operator evaluates expression, depending on the result, returns the value of one of the other two expressions. Accepts either three expressions in an ordered list or three named parameters. (**students**) |
| { $project: { \_id: 0, name: 1,  homeTown: { $ifNull: [ "$homeTown", "Unspecified" ] } } | Use 1st expression if not null, 2nd expression if 1st expression has a null result. (undefined or missing) |
| { $project: { \_id: 0, "student": 1, "average": { $avg: "$homework" },  "summary" : {  **$switch**: {  **branches**: [  { **case**: { $gte : [ { $avg : "$homework" }, 90 ] }  , **then**: "Doing great!" }  , { **case**: { $and : [ {$gte: [ { $avg : "$homework" }, 80 ]},  {$lt: [ { $avg : "$homework" }, 90 ]} ]}  , **then**: "Doing pretty well." }  , { **case**: { $gte : [ { $avg : "$homework" }, 0 ] }  , **then**: "Needs improvement." }  ]  , **default**: "No homework found."  }  } } } | Evaluates case expressions.  The 1st one to evaluate to true is executed  If none evaluate to true the default path is taken. Default is optional but required if a case path is not followed  (**scores**) |

|  |  |
| --- | --- |
| **Group Accumulator Operations** | |
| [$group](https://docs.mongodb.com/manual/reference/operator/aggregation/group/#pipe._S_group) stage operations, maintain state. Input is a single expression. All operations work in the $group stage  [$project](https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project) stage operations, do not maintain state. Input is one or multiple arguments. (if noted) | |
| { **$group**: { \_id: { $**year**: "$registrationDate"},  totalIncome: { $**sum**: "$income" },  averageIncome: { $**avg**: "$income" },  minIncome: { $**min**: "$income" },  stdDevPop: { $**stdDevPop**: "$income" },  firstIncome: { $**first**: "$income" },  count: { $**sum**: 1 } } } | Group students by registration year  sum of values ([$project](https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project) stage)  average of values (use $trunc to round) ([$project](https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project) stage)  lowest value (**$max**, highest) ([$project](https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project) stage)  population std dev (**$stdDevSamp**, sample) ([$project](https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project) stage)  value from first (**$last**, last) **document** in group  (**students**) |
| { **$group**: {  \_id: { year: { $year: "$registrationDate"} },  students: { **$push**: { name: "$name", major: "$major"}}}} | Returns an array of expression values for each group.  Group by year, output student's name and major.  **$addToSet** only adds the unique elements  (**students**) |

Collections based on the above examples can be found in:

<https://github.com/EdgeTechAcademy/MongoDB_Files/blob/master/students.mongo>

Some of the above examples deserve some additional attention

**students** collection. Here we are 'joining' the student collection with the courses collection. The courses collection has the readable name of the course.

1. **$unwind** the **grades** array in the students collection.
2. **$lookup**: match **grades.course** with **course** in the **courses** collection, name the sub-doc **CourseName**
3. **$lookup** returns an array, so **$unwind** it (not necessary but reduces the output to simple fields
4. Display the student name, course number and course name. give them nice names.
5. { **$unwind**: "**$grades**" }
6. , { **$lookup**:  
    { from: "**courses**"  
    , localField: "**grades.course**"  
    , foreignField: "course"  
    , as: "**CourseName**" } }
7. , { **$unwind**: "**$CourseName**"}
8. , { $project: { \_id: 0, name: 1  
    , CourseNum: "$grades.course"  
    , CourseName: "**$CourseName.name**"}}

**scores** collection. Homework, quizzes and extra credit are totaled. The students record is joined to get the student name. Grading is weighted: Homework at 60%, quizzes at 40% and extra credit is extra.

1. **$addFields** to total the **homework**, **quiz** and **extra** credit points
2. **$lookup:** Match **student** with **id** in the **students** collection. Name the sub-doc **StudentName**
3. **$unwind** the array returned by the lookup
4. **$addFields** to add a sub-document (final): student id and name, course #, raw totals and the weighted sum of the scores.  
   **$trunc** the totalScore as the math will create floating a point number
5. **$replaceRoot:** replace the document root with the final sub-document

1. { **$addFields**:  
    { **totalHomework**: { $sum: "**$homework**"}, **totalQuiz**: { $sum: "**$quizzes**"}   
    , **extraCredit**: { $sum: "**$extra**"} } }
2. , { **$lookup**:  
    { from: "**students**"  
    , localField: "**student**"  
    , foreignField: "**id**"  
    , as: "**StudentName**" } }
3. , {**$unwind**: "**$StudentName**"}
4. , { **$addFields**:  
    { **final**: { student: "$student"  
    , name: "**$StudentName**.name", course: "$course"  
    , "**Homework**": "**$totalHomework**", "**Quizzes**": "**$totalQuiz**"  
    , **extraCredit**:" **$extraCredit**"  
    , totalScore: { **$trunc**: { $add:  
    [ { $multiply: ["**$totalHomework**", 0.4 ] }  
    , { $multiply: ["**$totalQuiz**", 0.6 ] }  
    , "$extraCredit" ] }} } } }
5. , { **$replaceRoot**: { newRoot: "**$final**" } }